Rate Limiting in Linux

Version 1
06/30/2006

M. Bowden, M. Crawford
CD/CCF/Wide Area Systems

Why Rate Limit?

When a sender can transmit data faster than the network or receiver allows, packets are dropped. With the standard TCP control algorithm, this packet loss is guaranteed to occur and produces a sawtooth effect where the rate increases gradually until it exceeds the network capacity, drops by half, and increases again. On high bandwidth, high latency paths the recovery time for each cycle can be significant, leading to much lower average rates.

Splitting a single high bandwidth stream into a number of slower streams reduces this effect (mainly by adding several lower amplitude, out-of-phase sawtooth responses). Depending on the resonance however, some streams may take much longer to finish.

If the available network and receiver bandwidth is known to some extent, the stream (or streams) can be rate limited. Packet loss is avoided and there is no sawtooth effect. Of course, this is only useful if all streams sharing the path are rate limited and the total path bandwidth remains constant.

As an example, if an administrator wants to reserve most of the interface bandwidth for purpose X, but still allow some use for purpose Y, and if X and Y traffic can be distinguished by elements of the packet headers, then each flow can be rate limited to a specific fraction of the total bandwidth.

One method of controlling the rate is to route the traffic through separate output interfaces, with each interface limited to a certain maximum rate by the hardware. In some cases rate limiting can also be applied at the next stage router if the router software supports this.

A more flexible approach is to limit the rates in software at the source, using sstandard Linux commands.

Rate Limiting in Linux

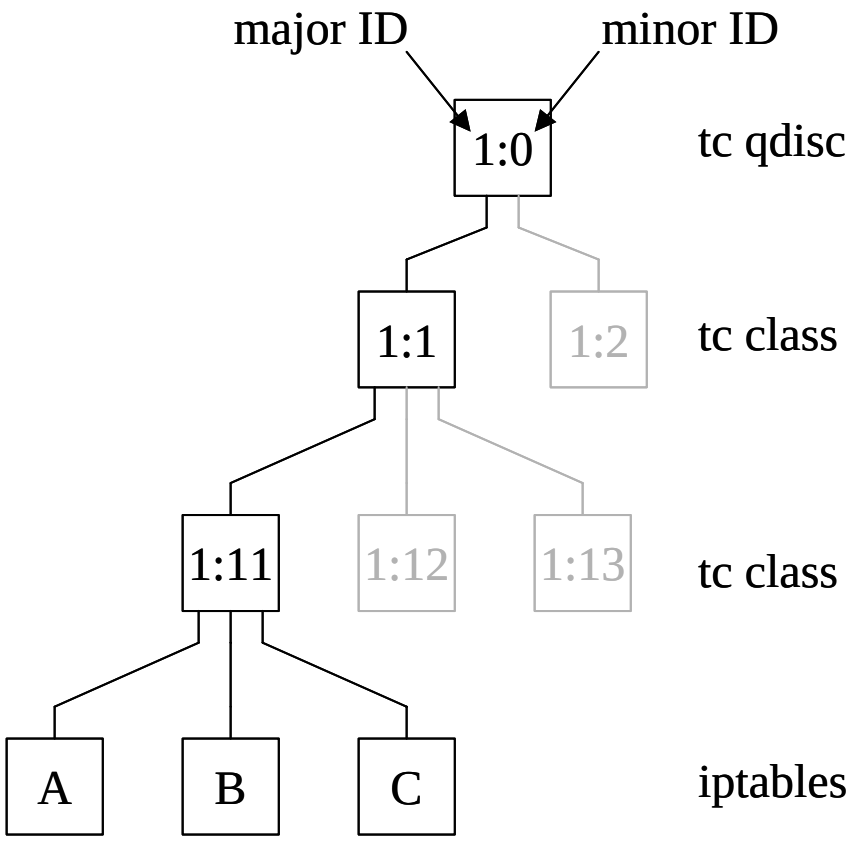
Two standard Linux commands for traffic control and packet filtering are used in the following examples;

tc – set up queues and place rate limits on data leaving each queue

iptables – define matching criteria for assigning packets to queues

For clarity, examples are given with command options in long form and without using shell variables. Only the basic rate limiting example has been tested, others may need to be corrected or modified for your application. The examples assume Linux version 2.4.18 (or higher).

The following illustration shows the basic queuing structure.



Each queuing discipline (qdisc) forms a tree with a common major ID. The top level minor ID is 0. All other minor IDs are arbitrary as long as they are unique. At the next level there may be a queue class (1:1 in the illustration) to set a total bandwidth limit for the qdisc. If no other qdiscs use the same interface, this level is not needed.

The available bandwidth is then divided among one or more queues (1:11, 1:12 and 1:13 in the illustration). One of these can be a default queue for any traffic that is not assigned to other queues. At the lowest level are the iptables filters (A,B,C in the illustration) which are used to assign packets to queues.

Example 1:

Four machines, each with a 1Gbps interface, are sending data to a common destination via a shared 1Gbps path with a long RTT. We would like to limit the output of each machine to 200Mbps for data packets sent to a specific IP address. Note: if there is other (non rate limited) traffic on the shared path, then rate limiting the output of these four machines does not necessarily prevent all packet loss.

Assuming the active interface is eth1 and the destination address is 1.2.3.4, each of the four machines would invoke a script similar to the following;

```
tc qdisc add dev eth1 root handle 1:0 htb default 12
tc class add dev eth1 parent 1:0 classid 1:1 htb rate 1000mbit
tc class add dev eth1 parent 1:1 classid 1:11 htb rate 200mbit \
   ceil 200mbit [mtu 18000]
tc class add dev eth1 parent 1:1 classid 1:12 htb rate 700mbit \
   ceil 700mbit [mtu 18000]
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 1.2.3.4 --jump CLASSIFY --set-class 1:11
```

Looking at each line individually,

```
tc qdisc add dev eth1 root handle 1:0 htb default 12
```

- create a root queuing discipline ("qdisc") for eth1 and assign it a handle of 1:0
- define it as a hierarchical token bucket queue ("htb")
- specify a default queue class (12) for any traffic not assigned to one of the other classes

```
tc class add dev eth1 parent 1:0 classid 1:1 htb rate 1000mbit
```

- create htb queue class (1:1) under the root qdisc (1:0) and limit it to the maximum rate of the interface (1000Mbps)

```
tc class add dev eth1 parent 1:1 classid 1:11 htb rate 200mbit \
   ceil 200mbit [mtu 18000]
```

- create htb queue subclass (under 1:1) with id 1:11 and limit it to a rate of 200Mbps.

```
tc class add dev eth1 parent 1:1 classid 1:12 htb rate 700mbit \
   ceil 700mbit [mtu 18000]
```

- create htb queue subclass (under 1:1) with id 1:12 and limit it to a rate of 700Mbps.
  (note: by default, all traffic not assigned to another subclass uses this subclass.)

```
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 1.2.3.4 --jump CLASSIFY --set-class 1:11
```

- match tcp packets with destination address 1.2.3.4 and classify these packets as belonging to queue 1:11

For simple matches (an IP address or subnet, port number, etc) you can use the "tc filter" command in place of the iptables command;

```
tc filter add dev eth1 parent 1:0 protocol ip prio 0 u32 \
   match ip dst 1.2.3.4 flowid 1:11
```

The only advantage to using "tc filter" is that it will automatically be deleted when you delete the base tc qdisc. Otherwise, "iptables" provides a much wider range of matching capabilities and is better documented.

If no other filtering is needed, then Example 1 can be simplified slightly;

```
tc qdisc add dev eth1 root handle 1:0 htb
tc class add dev eth1 parent 1:0 classid 1:1 htb rate 200mbit \
   ceil 200mbit [mtu 18000]
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 1.2.3.4 --jump CLASSIFY --set-class 1:1
```

Here the default (for unclassified traffic) is no queuing and no rate limiting, i.e. the packets are passed directly to the interface. Traffic with destination address 1.2.3.4 is rate limited by queue 1:1.

Example 2

To reduce problems associated with a high bandwidth-delay path, a 1Gbps transfer from a single source machine is split into ten streams (each stream assigned to a different port on the same destination machine). We rate limit each stream to 70Mbps, based on destination IP and port, with the remaining 300Mbps reserved for other traffic. In practice, the total real bandwidth will be less than 1000Mbps. The sum of all limits should reflect the real bandwidth.

```
tc qdisc add dev eth1 root handle 1:0 htb default 99
tc class add dev eth1 parent 1:0 classid 1:1 htb rate 1000mbit
tc class add dev eth1 parent 1:1 classid 1:10 htb rate 70mbit \
   ceil 70mbit [mtu 18000]
tc class add dev eth1 parent 1:1 classid 1:11 htb rate 70mbit \
   ceil 70mbit [mtu 18000]
tc class add dev eth1 parent 1:1 classid 1:12 htb rate 70mbit \
   ceil 70mbit [mtu 18000]
tc class add dev eth1 parent 1:1 classid 1:13 htb rate 70mbit \
   ceil 70mbit [mtu 18000]
tc class add dev eth1 parent 1:1 classid 1:14 htb rate 70mbit \
   ceil 70mbit [mtu 18000]
tc class add dev eth1 parent 1:1 classid 1:15 htb rate 70mbit \
   ceil 70mbit [mtu 18000]
tc class add dev eth1 parent 1:1 classid 1:16 htb rate 70mbit \
   ceil 70mbit [mtu 18000]
tc class add dev eth1 parent 1:1 classid 1:17 htb rate 70mbit \
   ceil 70mbit [mtu 18000]
tc class add dev eth1 parent 1:1 classid 1:18 htb rate 70mbit \
   ceil 70mbit [mtu 18000]
tc class add dev eth1 parent 1:1 classid 1:19 htb rate 70mbit \
   ceil 70mbit [mtu 18000]
tc class add dev eth1 parent 1:1 classid 1:99 htb rate 300mbit \
   ceil 300mbit [mtu 18000]
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 1.2.3.4 --destination-port 1000 \
   --jump CLASSIFY --set-class 1:10
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 1.2.3.4 --destination-port 1001 \
   --jump CLASSIFY --set-class 1:11
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 1.2.3.4 --destination-port 1002 \
   --jump CLASSIFY --set-class 1:12
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 1.2.3.4 --destination-port 1003 \
   --jump CLASSIFY --set-class 1:13
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 1.2.3.4 --destination-port 1004 \
   --jump CLASSIFY --set-class 1:14
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 1.2.3.4 --destination-port 1005 \
   --jump CLASSIFY --set-class 1:15
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 1.2.3.4 --destination-port 1006 \
   --jump CLASSIFY --set-class 1:16
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 1.2.3.4 --destination-port 1007 \
   --jump CLASSIFY --set-class 1:17
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 1.2.3.4 --destination-port 1008 \
   --jump CLASSIFY --set-class 1:18
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 1.2.3.4 --destination-port 1009 \
   --jump CLASSIFY --set-class 1:19
```

Example 3

A web/FTP server saves at least 60% of the available bandwidth for on-site traffic, but if there are no on-site clients, then off-site clients can use 100%.

```
tc qdisc add dev eth1 root handle 1:0 htb
tc class add dev eth1 parent 1:0 classid 1:1 htb rate 600mbit \
   ceil 1000mbit [mtu 18000]
tc class add dev eth1 parent 1:0 classid 1:2 htb rate 400mbit \
   ceil 1000mbit [mtu 18000]
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 131.225.0.0/16 --source-port 80 \
   --jump CLASSIFY --set-class 1:1
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 131.225.0.0/16 --source-port 20:21 \
   --jump CLASSIFY --set-class 1:1
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination 131.225.0.0/16 --source-port 1024:5000 \
   --jump CLASSIFY --set-class 1:1
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination !131.225.0.0/16 --source-port 80 \
   --jump CLASSIFY --set-class 1:2
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination !131.225.0.0/16 --source-port 20:21 \
   --jump CLASSIFY --set-class 1:2
iptables --table mangle --append POSTROUTING --out-interface eth1 \
   --protocol tcp --destination !131.225.0.0/16 --source-port 1024:5000 \
   --jump CLASSIFY --set-class 1:2
```

In this example, all traffic from ports 20, 21, 80, and 1024-5000 destined for IP subnet 131.225 (on-site) is limited according to queue 1:1. Traffic from the same ports, but destined for any other IP address (off-site) is limited according to queue 1:2. Traffic from all other ports (on or off-site) is not limited. Note that the only difference in the on-site/off-site filter definitions is the "!" before the IP address to invert the range. The rate ceiling for both queues is set to the interface maximum so that either queue can use the full bandwidth in the absence of traffic in the other queue.

mtu Parameter

The "mtu" parameter in the tc class definition is needed if the interface uses jumbo packets. The htb queue assumes a default packet size of 1600 if mtu is not specified. Check the current interface mtu and qdisc type with

```
ip link list
```

Note: the mtu for jumbo packets is typically 9000 Bytes, but if the "mtu" parameter is set to 9000, tests using NDT show a rate approximately 25% higher than the limit requested. With the mtu parameter set to 18000, the rate reported by NDT is much closer to the requested rate. We recommend setting mtu to 18000 when using jumbo packets. For standard packets, the default mtu is satisfactory.

rate and ceil Parameters

The "rate" parameter sets the desired *average* output rate. The "ceil" parameter sets the *maximum* output rate. Setting "ceil" equal to "rate" provides a fixed limit, which then reserves remaining bandwidth for other traffic. If "ceil" is higher than "rate", the queue is free to use any bandwidth not being used by other traffic (up to the "ceil" limit).

Note: tc rate parameters use "mbit" to designate "megabits per second" and "mbps" to designate "megaBytes per second" ...these are easily confused. Also, the documentation states that a number without a qualifier is in Bytes per second, but it appears that it is actually in bits per second.

Viewing/Undoing the Damage

The queues and filters defined by tc and iptables remain in effect until deleted. Since they are applied at the root level, they should be removed when no longer needed. This section tells you how to undo all traffic classification and rate control.

To view the current iptables rules,

```
iptables --table mangle --list POSTROUTING --verbose
```

To delete the iptables rules,

```
iptables --table mangle --delete POSTROUTING 1
```
(remove first [or nth] rule in list)

```
iptables --table mangle --flush POSTROUTING
```
(remove all rules in list)

To view the current tc qdisc and classes,

```
tc qdisc show dev eth1

tc class show dev eth1
```

(if no queue is implemented, the default qdisc will be "pfifo_fast")

To delete the tc root queue and all subclasses,

```
tc qdisc del dev eth1 root
```

iptables Filtering

This section is a very abbreviated version of the iptables man page (March 9,2002). Refer to the complete man page for options not shown here.

SYNOPSIS

iptables --table mangle --append POSTROUTING [filter_parameters] \
--jump CLASSIFY --set-class tc_class_id

FILTER_PARAMETERS

The following parameters make up a rule specification.

-p, --protocol [!] *protocol*
The protocol of the rule or of the packet to check. The specified protocol can be one of *tcp*, *udp*, *icmp*, or *all*, or it can be a numeric value, representing one of these protocols or a different one. A protocol name from /etc/protocols is also allowed. A "!" argument before the protocol inverts the test. The number zero is equivalent to *all*. Protocol *all* will match with all protocols and is taken as default when this option is omitted.

-s, --source [!] *address*[/*mask*]
Source specification. *Address* can be either a network name, a hostname (please note that specifying any name to be resolved with a remote query such as DNS is a really bad idea), a network IP address (with /mask), or a plain IP address. The *mask* can be either a network mask or a plain number, specifying the number of 1's at the left side of the network mask. Thus, a mask of *24* is equivalent to *255.255.255.0*. A "!" argument before the address specification inverts the sense of the address. The flag **--src** is an alias for this option.

-d, --destination [!] *address*[/*mask*]
Destination specification. See the description of the **-s** (source) flag for a detailed description of the syntax. The flag **--dst** is an alias for this option.

-o, --out-interface [!] *name*
Name of an interface via which a packet is going to be sent. When the "!" argument is used before the interface name, the sense is inverted. If the interface name ends in a "+", then any interface which begins with this name will match. If this option is omitted, any interface name will match.

[!] -f, --fragment
This means that the rule only refers to second and further fragments of fragmented packets. Since there is no way to tell the source or destination ports of such a packet (or ICMP type), such a packet will not match any rules which specify them. When the "!"

argument precedes the "-f" flag, the rule will only match head fragments, or unfragmented packets.

MATCH EXTENSIONS

iptables can use extended packet matching modules. These are loaded in two ways: implicitly, when **-p** or **--protocol** is specified, or with the **-m** or **--match** options, followed by the matching module name; after these, various extra command line options become available, depending on the specific module. You can specify multiple extended match modules in one line, and you can use the **-h** or **--help** options after the module has been specified to receive help specific to that module.
The following are included in the base package, and most of these can be preceded by a **!** to invert the sense of the match.

addrtype

This module matches packets based on their **address type.** Address types are used within the kernel networking stack and categorize addresses into various groups. The exact definition of that group depends on the specific layer three protocol.
The following address types are possible:
UNSPEC an unspecified address (i.e. 0.0.0.0)
UNICAST an unicast address
LOCAL a local address
BROADCAST a broadcast address
ANYCAST an anycast packet
MULTICAST a multicast address
BLACKHOLE a blackhole address
UNREACHABLE an unreachable address
PROHIBIT a prohibited address

--dst-type *type*
Matches if the destination address is of given type

ah

This module matches the SPIs in AH header of IPSec packets.
--ahspi [!] *spi*[:*spi*]

childlevel

This is an experimental module. It matches on whether the packet is part of a master connection or one of its children (or grandchildren, etc). For instance, most packets are level 0. FTP data transfer is level 1.
--childlevel [!] *level*

comment

Allows you to add comments (up to 256 characters) to any rule.
--comment *comment*
Example:
iptables -A INPUT -s 192.168.0.0/16 -m comment --comment "A privatized IP block"

condition

This matches if a specific /proc filename is '0' or '1'.
--condition *[!] filename*
Match on boolean value stored in /proc/net/ipt_condition/filename file

connrate

This module matches the current transfer rate in a connection.
--connrate *[!] [from*]:[*to*]
Match against the current connection transfer rate being within 'from' and 'to' bytes per second. When the "!" argument is used before the range, the sense of the match is inverted.

dscp

This module matches the 6 bit DSCP field within the TOS field in the IP header. DSCP has superseded TOS within the IETF.
--dscp *value*
Match against a numeric (decimal or hex) value [0-32].
--dscp-class *DiffServ Class*
Match the DiffServ class. This value may be any of the BE, EF, AFxx or CSx classes. It will then be converted into it's according numeric value.

ecn

This allows you to match the ECN bits of the IPv4 and TCP header. ECN is the Explicit Congestion Notification mechanism as specified in RFC3168
--ecn-tcp-cwr
This matches if the TCP ECN CWR (Congestion Window Reduced) bit is set.
--ecn-tcp-ece
This matches if the TCP ECN ECE (ECN Echo) bit is set.
--ecn-ip-ect *num*

This matches a particular IPv4 ECT (ECN-Capable Transport). You have to specify a number between `0' and `3'.

esp

This module matches the SPIs in ESP header of IPSec packets.
--espspi [!] *spi*[:*spi*]

icmp

This extension is loaded if `--protocol icmp' is specified. It provides the following option:
--icmp-type [!] *typename*
This allows specification of the ICMP type, which can be a numeric ICMP type, or one of the ICMP type names shown by the command
iptables -p icmp -h

iprange

This matches on a given arbitrary range of IPv4 addresses
[!]--*src-range* **ip-ip**
Match source IP in the specified range.
[!]--*dst-range* **ip-ip**
Match destination IP in the specified range.

length

This module matches the length of a packet against a specific value or range of values.
--length *length*[:*length*]

mport

This module matches a set of source or destination ports. Up to 15 ports can be specified. It can only be used in conjunction with **-p tcp** or **-p udp**.
--source-ports *port*[,*port*[,*port*...]]
Match if the source port is one of the given ports. The flag **--sports** is a convenient alias for this option.
--destination-ports *port*[,*port*[,*port*...]]
Match if the destination port is one of the given ports. The flag **--dports** is a convenient alias for this option.
--ports *port*[,*port*[,*port*...]]

Match if the both the source and destination ports are equal to each other and to one of the given ports.

multiport

This module matches a set of source or destination ports. Up to 15 ports can be specified. It can only be used in conjunction with **-p tcp** or **-p udp**.

--source-ports *port*[,*port*[,*port*...]]
Match if the source port is one of the given ports. The flag **--sports** is a convenient alias for this option.

--destination-ports *port*[,*port*[,*port*...]]
Match if the destination port is one of the given ports. The flag **--dports** is a convenient alias for this option.

--ports *port*[,*port*[,*port*...]]
Match if the both the source and destination ports are equal to each other and to one of the given ports.

owner

This module attempts to match various characteristics of the packet creator, for locally-generated packets. Also note that some packets (such as ICMP ping responses) may have no owner, and hence never match.

--uid-owner *userid*
Matches if the packet was created by a process with the given effective user id.

--gid-owner *groupid*
Matches if the packet was created by a process with the given effective group id.

--pid-owner *processid*
Matches if the packet was created by a process with the given process id.

--sid-owner *sessionid*
Matches if the packet was created by a process in the given session group.

--cmd-owner *name*
Matches if the packet was created by a process with the given command name. (this option is present only if iptables was compiled under a kernel supporting this feature)

NOTE: pid, sid and command matching are broken on SMP

pkttype

This module matches the link-layer packet type.

--pkt-type *[unicast|broadcast|multicast*]

policy

This modules matches the policy used by IPsec for handling a packet.

--dir *out*
out is valid in the **POSTROUTING** chain.
--pol *none|ipsec*
Matches if the packet is subject to IPsec processing.
--strict
Selects whether to match the exact policy or match if any rule of the policy matches the given policy.
--reqid *id*
Matches the reqid of the policy rule. The reqid can be specified with **setkey** using **unique:id** as level.
--spi *spi*
Matches the SPI of the SA.
--proto *ah|esp|ipcomp*
Matches the encapsulation protocol.
--mode *tunnel|transport*
Matches the encapsulation mode.
--tunnel-src *addr[/mask]*
Matches the source address of a tunnel. Only valid with --mode tunnel.
--tunnel-dst *addr[/mask]*
Matches the destination address of a tunnel. Only valid with --mode tunnel.
--next
Start the next element in the policy specification. Can only be used with --strict

set

This modules matches IP sets which can be defined by ipset.
--set setname flag[,flag...]
where flags are **src** and/or **dst** and there can be no more than six of them.

state

This module, when combined with connection tracking, allows access to the connection tracking state for this packet.
--state *state*
Where state is a comma separated list of the connection states to match. Possible states are **INVALID** meaning that the packet could not be identified for some reason which includes running out of memory and ICMP errors which don't correspond to any known connection, **ESTABLISHED** meaning that the packet is associated with a connection which has seen packets in both directions, **NEW** meaning that the packet has started a new connection, or otherwise associated with a connection which has not seen packets in both directions, and **RELATED** meaning that the packet is starting a new connection, but is associated with an existing connection, such as an FTP data transfer, or an ICMP error.

tcp

These extensions are loaded if `--protocol tcp' is specified. It provides the following options:

--source-port [!] *port*[:*port*]

Source port or port range specification. This can either be a service name or a port number. An inclusive range can also be specified, using the format *port*:*port*. If the first port is omitted, "0" is assumed; if the last is omitted, "65535" is assumed. If the second port greater then the first they will be swapped. The flag **--sport** is a convenient alias for this option.

--destination-port [!] *port*[:*port*]

Destination port or port range specification. The flag **--dport** is a convenient alias for this option.

--tcp-flags [!] *mask comp*

Match when the TCP flags are as specified. The first argument is the flags which we should examine, written as a comma-separated list, and the second argument is a comma-separated list of flags which must be set. Flags are: **SYN ACK FIN RST URG PSH ALL NONE**.

[!] --syn

Only match TCP packets with the SYN bit set and the ACK and RST bits cleared. Such packets are used to request TCP connection initiation; for example, blocking such packets coming in an interface will prevent incoming TCP connections, but outgoing TCP connections will be unaffected. It is equivalent to **--tcp-flags SYN,RST,ACK SYN**. If the "!" flag precedes the "--syn", the sense of the option is inverted.

--tcp-option [!] *number*

Match if TCP option set.

--mss *value*[:*value*]

Match TCP SYN or SYN/ACK packets with the specified MSS value (or range), which control the maximum packet size for that connection.

tcpmss

This matches the TCP MSS (maximum segment size) field of the TCP header. You can only use this on TCP SYN or SYN/ACK packets, since the MSS is only negotiated during the TCP handshake at connection startup time.

[!] --mss *value[:value]*

Match a given TCP MSS value or range.

tos

This module matches the 8 bits of Type of Service field in the IP header (ie. including the precedence bits).

--tos *tos*

The argument is either a standard name, (use iptables -m tos -h to see the list), or a numeric value to match.

ttl

This module matches the time to live field in the IP header.
--ttl-eq *ttl*
Matches the given TTL value.
--ttl-gt *ttl*
Matches if TTL is greater than the given TTL value.
--ttl-lt *ttl*
Matches if TTL is less than the given TTL value.

udp

These extensions are loaded if `--protocol udp' is specified. It provides the following options:
--source-port [!] *port*[:*port*]
Source port or port range specification. See the description of the **--source-port** option of the TCP extension for details.
--destination-port [!] *port*[:*port*]
Destination port or port range specification. See the description of the **--destination-port** option of the TCP extension for details.